A Software Quality Model and Metrics For Risk Assessment

L. Hyatt, NASA/Goddard Space Flight Center, Code 302, Greenbelt MD, USA 20771. Tel: 301-2867475 Fax: 301-2861701

L. Rosenberg, Ph.D., Unisys/Goddard Space flight Center, Code 300.1, Greenbelt MD, USA 20771. Tel: 301-2860087, Fax: 301-2860304

ABSTRACT

This paper defines a Software Quality Model and associated attributes and then uses the model as a basis for a discussion of risk. Specific quality goals and attributes are selected based on their importance to a software development project and their ability to be quantified. Risks that can be determined by the model's metrics are identified. A core set of metrics relating to the software development process and its products is defined. Measurements for each metric and their usability and applicability discussed.

1.0 INTRODUCTION

The National Aeronautics and Space Administration (NASA) is increasingly reliant on software for the functionality of the systems it develops and uses. The Agency has recognized the importance of improving the way it develops software, and has adopted a software strategic plan to guide the improvement process. At the Goddard Space Flight Center (GSFC), the implementation of the plan has led to the establishment of the Software Assurance Technology Center (SATC). As part of its role of improving the quality of software at GSFC and NASA the SATC has software metrics as one of its areas of emphasis.

2.0 MANAGER'S QUALITY PERSPECTIVE

In the software development process at GSFC, while quality is seen from many perspectives, the most important view is that of the project manager. The situation in which the software is developed and used determines the project manager's view. In the process flow for building a space mission, the software is integral to developing the system. For example, flight software and the spacecraft test system software must be completed at least to some level of functionality before the integration and test of the spacecraft can be begun. Similar dependencies exist in other parts of the system.

Thus the project manager's view of software quality is pragmatic and relatively simple - high quality software is software that "works well enough" to serve its intended function and is "available when needed" to perform that function. The criterion of "Works Well Enough" includes satisfaction of functional, performance, and interface requirements as well as the satisfaction of typical "ility" requirements such as reliability, maintainability, reusability and correctness. The criterion of "Available When Needed" is dependent upon the software's role in the system. Delays in availability of some software could delay the whole system and postpone the most critical date of all - the launch date.

3.0 SATC QUALITY MODEL

As part of its mission to improve the quality of NASA software, the SATC is assisting software managers in establishing metrics programs and in interpreting the resulting metrics. Using the results of these metric programs and discussions with projects as a basis, the SATC is currently defining and testing a quality model for software. Our experience indicates that the project manager's pragmatic view of software quality cannot be evaluated using only software product goals and attributes; goals and attributes for the development process through the life cycle must also be evaluated. Questions of interest are of the nature:

Can we identify early in the development process the risks to successful completion of the software when it is needed?

Will we achieve adequate correctness / reliability with the test resources and schedule currently allocated?

What sections of the code are of the highest risk for reliability, maintainability and reuse?

ISO 9126 is a typical quality model. It consists of a set of goals, each defined by a set of attributes. The attributes are assessed by a set of metrics. Following the ISO structure, the SATC model defines a set of goals that are important at GSFC. The goals are then related to software product and process attributes that allow indications of the probability of success in meeting the goals. A set of metrics is chosen or developed that measures the attributes. The goals relate to the project manager's questions about "working well enough" and being "available on time." From this pragmatic description of software quality, we derive four goals:

Requirements Quality

Product Quality

Implementation Effectivity

Testing Effectivity

The model's goals must be capable of being evaluated by a set of attributes that help to define and classify risks. The attributes must be "measurable" by a set of metrics that is possible to collect within the confines of the software development process and that will yield the desired information.

4.0 REQUIREMENTS QUALITY

The objectives for this first goal are complete, unambiguous, and understandable requirements, the stabilization of requirements as quickly as possible, and the traceability of all requirements from their source to the software requirements and then through design to implementation and test. The associated attributes are:

Ambiguity - Potential multiple meanings.

Completeness - Items left to be specified.

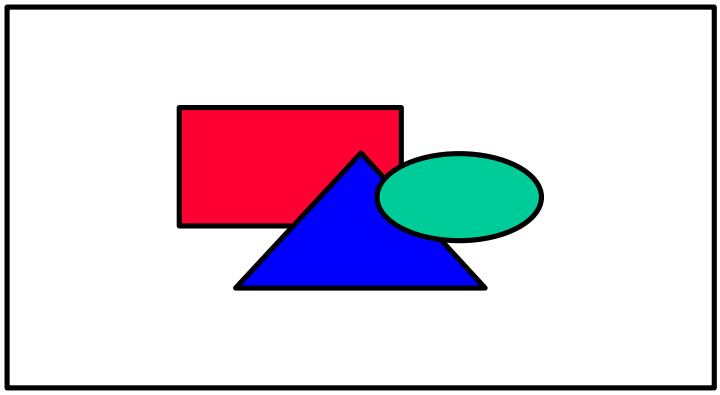
Understandability - The readability of the document.

Requirement Volatility - The rate and the life cycle phase when requirements changes are made.

Tracability - The tracability of the requirements upward and downward.

4.1 Requirements Attributes and Metrics

Ambiguous requirements are those that may have multiple meanings or those that seem to leave to the developer the decision whether or not to implement a requirement. There are two metrics used to evaluate the ambiguity of the requirements document: the number of weak phases and the number of optional phrases, as shown in the following table.



Ambiguity is computed as the sum of the count of weak phrases and the count of optional phrases.

If the requirements document is complete, it will have all of the requirements specified in adequate detail to allow design and implementation to proceed. The metrics used to evaluate completeness are a count of the TBA (to be added) and TBD (to be determined) acronyms used in the document. These acronyms are those most often used to indicate that a portion of the requirements must be supplied at some future time.

The attribute of understandably relates to the ability of the developers to understand clearly what is met by the requirements document. At this time we are looking at two metrics, one based on numbering structure and a second based on readability evaluations. However, the two metrics are difficult to verify as having an impact on the risk of using the document and therefore the attribute of understandability cannot be heavily used in evaluating overall requirements quality.

Volatility measures the rate of change of requirements. The impact of changes to the requirements increases as one gets closer to the time when the software is to be released. The metrics for volatility are then the percentage of requirements changed in a given time period, computed as number of requirements in the changes to the document divided by the base count of requirements. The changes to be assessed may come from the project configuration management system, or the changes can be located by comparisons of successive versions of the document.

Software requirements must be traceable to system requirements to assure that the software when developed will work properly in the system setting. The software requirements must also be traceable to the implementing design and code, to ensure that they are in fact implemented. They

must be traceable to tests to ensure that they have been validated and verified. There are two metrics for tracability, the number of requirements not traced to higher level requirements and the number not traced to code and tests. To compute these metrics requires a trace matrix which contains the trace information.

4.2 Requirements Risk

It is generally accepted that poorly written, rapidly changing requirements are a principal source of project risk (and indeed, of project failure). The SATC is working on methods to measure requirements document quality in much the same way that code quality is measured, that is, by measuring characteristics of the document itself.

Volatility is an important factor in risk, and extensive measures are often taken to reduce its impact. The later in the life cycle changes are made to requirements, the more resources takes to implement them. The earlier in the life cycle the requirements stabilize, the less the risk.

Overall assessment of requirements risk has to blend the impact of all of the requirements metrics. To reduce risk, specific requirements that are ambiguous or that contain TBAs or TBDs can be identified and revised. It should be noted that requirements risk must be measured throughout the life cycle.

5.0 PRODUCT QUALITY

An important objective of a software development project is to develop code and documentation that will meeting the project's "ility" requirements. The specific attributes for product quality are:

Structure/Architecture - The evaluation of the constructs within a module to identify possible error-prone modules and to indicate potential problems in usability and maintainability.

Reuse - The suitability of the software for reuse in a different context or application.

Maintainability - The suitability of the software for ease of locating and fixing a fault in the program.

Documentation - The adequacy of internal code documentation and external documentation.

5.1 Product Quality Attributes and Metrics

The attributes of Structure/Architecture, Reusability and Maintainability use the same metrics, but with different emphasis. The metrics are complexity, size, and the correlation of complexity with size.

It is generally accepted that more complex modules are more difficult to understand and have a higher probability of defects than less complex modules. Thus complexity has a direct impact on overall quality and specifically on maintainability . Projects developing code intended for reuse should be even more concerned with complexity since it may later be found to be more expedient to rewrite highly complex modules than to reuse them. While there are many different types of complexity measurements, the one used by the SATC is logical (Cyclomatic) complexity, which is computed as the number of linearly independent test paths.

Size is one of the oldest and most common forms of software measurement. Size of modules is itself a quality indicator. General industry standards suggest that 50 to 100 lines is the maximum size that any module should attain; larger modules tend to be difficult to understand and thus lower in maintainability and reusability.

While both size and complexity are useful metrics, the correlation of the two produces even more information. The SATC has found that much of the code produced at GSFC has a linear relationship between complexity and size, such that the complexity of a "normal" module can be predicted by its size. Those modules above the linear relationship - those that are more than normally complex - are the ones whose quality is suspect.

Documentation is the description of the content of the code. The metric for this attribute is the percentage of comments within the code. Comment percentage is calculated by dividing the number of comments by the total lines of code less the blank lines.

5.2 Product Quality Risk

While poor quality of any product is a risk to the specific objectives of the project, some of the best measures of risk come from correlations of the base metrics. The SATC uses a scatter plot of the complexity versus the size of all of the code modules in a program or segment of the system software. The scatter plot identifies the risks to correctness and, especially, maintainability and reusability. Risk are assigned to areas in the scatter plot based on the SATC's experience.

The risk determined by the complexity - size correlation can be combined with the commenting percentage for high risk modules to identify code that should be improved.

6.0 IMPLEMENTATION EFFECTIVITY

The objective of implementation effectivity is to maximize the effectiveness of resources within the project scheduled activities. The attributes of this goal are:

Resource use - The extent resource usage is appropriate for the phase of the project.

Completion Rates - progress made in completing items such as peer reviews, or turnover of completed modules to CM.

6.1 Implementation Effectivity Attributes and Metrics

The resource metric used by the SATC is personnel hours devoted to a set of life cycle activities. The measures collected are staff hours spent on a list of activities that is tailored for the project (and may have to change as the project progresses). Key activities that are measured include: requirements analysis, coding, testing, corrective action, training, and others.

Completion rates of tasks and product components are a good indicator of risk that a project will or will not be able to provide products on the needed schedule. Completion rates can be measured, and if a detailed schedule is available, completions can be compared to the schedule to provide planned versus actual charts. This, combined with the resource measures, gives a good picture of what is going on and the impact of the carryover of prior phase activities.

6.2 Implementation Effectivity and Risk

The risks that can be determined from the metrics of resource use are based on the appropriateness of the tasks to which resources are being applied at a given time during the life cycle. To the extent that those activities do not match the expected or planned activities, an element of risk may have been identified. For example, work on prior phase activities during the current phase is a risk indicator. If significant effort is being expended on requirements activities during the design phase (or worse, during the implementation phase), there is significant risk that the project will not be able to meet its schedule objective.

As an example of the use of completion rate data in determining risks, the completion rate of tests can be used to estimate the risk of competing all tests on time. The risk is measured by use of test report data. By determining the rate of completion of tests a simple calculation can show if the remaining tests can be completed in the time available.

7.0 TESTING EFFECTIVITY

The objectives for effective testing is to locate and repair faults in the software, to identify error-prone software, and to complete testing on schedule with sufficient faults found and repaired that the software will operate "well enough" when it is put into operation. The attribute measured is:

Correctness - the number of faults remaining.

7.1 Testing Effectivity Attributes and Metrics

Correctness is defined as the extent the code fulfills specifications. One implication is that the software must be error free. Measurements of errors is only recorded during system level tests.

The SATC has developed a model that uses the typical cumulative error curve equation and does a curve fit with the error data from testing. After the projections have settled down (about a third of the way through the test process), the fitted curve can be used to predict the total number of errors that will be found and the schedule resource needed to find them. We have also have had some success in predicting the total number of errors by criticality category.

Correctness is reported as the number of errors found and, after the model has settled down, the expected total number of errors that will be found.

7.2 Testing Effectivity Risk

The correctness measurements lend themselves to a number of risk projections. If the project has quantified the percentage of the errors that are to be found before the software is accepted, the model can be used to project when that will happen. If the time is significantly after the time at which the schedule requires the software to be available, then risk is high.

Another risk determination can be done by looking at the code modules or sub-systems in which the errors occur. Segments of code with more than the average number of errors may well need another look to see if re-engineering should be considered. Also, the time to fix errors should be tracked. If this is increasing, then there is a risk to correctness and to schedule.

Finally, the location of faults that caused high criticality errors should be tracked. A

concentration of them in a segment of code indicates a risk that the requirements are not well understood, or that the design is not suitable.

8.0 MULTI - METRIC RISK EVALUATIONS

In addition to the risks that can be determined and classified based on the metrics collected for a specific goal, some risk determinations can be improved by consideration of metrics from another goal. For example, high requirements volatility can not only be measured by a count of requirements changes, but by the amount of staff resources being spent on requirements after the end of the requirements analysis phase of the life cycle.

Another multiple metric risk determination can be done by using the risk ranking for modules based on their size and complexity and adding consideration of the number and criticality of errors found in the modules. As is obvious, modules with high values in all areas are those that should be examined for re-engineering.

9.0 SUMMARY AND FUTURE WORK

The SATC has applied some of the concepts from the traditional models of software quality to develop a unique model that fits the needs of project managers in the NASA and GSFC environment. The model is dynamic, not static, in that it supports projections about specific project risks at project milestones. The model uses a broad range of measures, since it contains goals, attributes, and metrics for both software products and development processes.

New metrics, especially those relating to requirements quality, need to be validated, and a set of higher level design quality metrics is needed to supplement those done at the code level. Our long term objective is to be able to establish a numerical metric scale for assessment of all of the metrics that affect software quality.

Presented at European Space Agency Software Assurance Symposium, Netherlands, March, 1996 and the 8th Annual Software Technology Conference, Utah, April, 1996.